# Contents

# 1 Setting

## 1.1 vimrc

```
1 set nocp ai si nu et bs=2 mouse=a
2 set ts=2 sts=2 sw=2 hls showmatch
3 set ruler rulerformat=%17.(%l:%c%)
4 set noswapfile autoread wildmenu wildmode=list:longest
5 syntax on | colorscheme evening
6
7 map <F5> <ESC>:w<CR>:!g++ -g -Wall --std=c++0x -O2 %:r.cpp -o %:r && %:r < %:r.
    in > %:r.out<CR>
8 map <F6> <ESC>:w<CR>:!g++ -g -Wall --std=c++0x -O2 %:r.cpp -o %:r && %:r < %:r.
    in<CR>
9
10 map k gk
11 map j gj
12
13 map <C-h> <C-w>h
14 map <C-j> <C-w>j
```

```
15 map <C-k> <C-w>k
16 map <C-l> <C-w>l
17
18 map <C-t> :tabnew<CR>
19
20 command -nargs=1 PS :cd d:/ | :vi <args>.cpp | vs <args>.in | sp <args>.out
```

# 2   Math

## 2.1   Basic Arithmetic

```
1 typedef long long ll;
2 typedef unsigned long long ull;
3
4 // calculate lg2(a)
5 inline int lg2(ll a)
6 {
7     return 63 - __builtin_clzll(a);
8 }
9
10 // calculate the number of 1-bits
11 inline int bitcount(ll a)
12 {
13     return __builtin_popcountll(a);
14 }
15
16 // calculate ceil(a/b)
17 // |a|, |b| <= (2^63)-1 (does not dover -2^63)
18 ll ceildiv(ll a, ll b) {
19     if (b < 0) return ceildiv(-a, -b);
20     if (a < 0) return (-a) / b;
21     return ((ull)a + (ull)b - 1ull) / b;
22 }
23
24 // calculate floor(a/b)
25 // |a|, |b| <= (2^63)-1 (does not cover -2^63)
26 ll floordiv(ll a, ll b) {
27     if (b < 0) return floordiv(-a, -b);
28     if (a >= 0) return a / b;
29     return -(ll)(((ull)(-a) + b - 1) / b);
30 }
31
32 // calculate a*b % m
33 // x86-64 only
34 ll large_mod_mul(ll a, ll b, ll m)
35 {
36     return ll((__int128)a*(__int128)b%m);
37 }
38
39 // calculate a*b % m
40 // |m| < 2^62, x86 available
41 // O(logb)
42 ll large_mod_mul(ll a, ll b, ll m)
43 {
44     a %= m; b %= m; ll r = 0, v = a;
45     while (b) {
46         if (b&1) r = (r + v) % m;
47         b >>= 1;
48         v = (v << 1) % m;
49     }
50     return r;
51 }
52
53 // calculate n^k % m
54 ll modpow(ll n, ll k, ll m) {
55     ll ret = 1;
56     n %= m;
57     while (k) {
58         if (k & 1) ret = large_mod_mul(ret, n, m);
59         n = large_mod_mul(n, n, m);
60         k /= 2;
61     }
62     return ret;
63 }
64
65 // calculate gcd(a, b)
66 ll gcd(ll a, ll b) {
67     return b == 0 ? a : gcd(b, a % b);
68 }
69
70 // find a pair (c, d) s.t. ac + bd = gcd(a, b)
71 pair<ll, ll> extended_gcd(ll a, ll b) {
72     if (b == 0) return { 1, 0 };
73     auto t = extended_gcd(b, a % b);
74     return { t.second, t.first - t.second * (a / b) };
75 }
76
77 // find x in [0,m) s.t. ax === gcd(a, m) (mod m)
78 ll modinverse(ll a, ll m) {
79     return (extended_gcd(a, m).first % m + m) % m;
80 }
81
82 // calculate modular inverse for 1 ~ n
83 void calc_range_modinv(int n, int mod, int ret[]) {
84     ret[1] = 1;
85     for (int i = 2; i <= n; ++i)
86         ret[i] = (ll)(mod - mod/i) * ret[mod%i] % mod;
87 }
```

## 2.2   Sieve Methods : Prime, Divisor, Euler phi

```
1 // find prime numbers in 1 ~ n
2 // ret[x] = false  ->  x is prime
3 // O(n*loglogn)
4 void sieve(int n, bool ret[]) {
5     for (int i = 2; i * i <= n; ++i)
6         if (!ret[i])
```

```
 7            for (int j = i * i; j <= n; j += i)
 8                ret[i] = true;
 9 }
10
11 // calculate number of divisors for 1 ~ n
12 // when you need to calculate sum, change += 1 to += i
13 // O(n*logn)
14 void num_of_divisors(int n, int ret[]) {
15     for (int i = 1; i <= n; ++i)
16         for (int j = i; j <= n; j += i)
17             ret[j] += 1;
18 }
19
20 // calculate euler totient function for 1 ~ n
21 // phi(n) = number of x s.t. 0 < x < n && gcd(n, x) = 1
22 // O(n*loglogn)
23 void euler_phi(int n, int ret[]) {
24     for (int i = 1; i <= n; ++i) ret[i] = i;
25     for (int i = 2; i <= n; ++i)
26         if (ret[i] == i)
27             for (int j = i; j <= n; j += i)
28                 ret[j] -= ret[j] / i;
29 }
```

## 2.3   Primality Test

```
 1 bool test_witness(ull a, ull n, ull s) {
 2     if (a >= n) a %= n;
 3     if (a <= 1) return true;
 4     ull d = n >> s;
 5     ull x = modpow(a, d, n);
 6     if (x == 1 || x == n-1) return true;
 7     while (s-- > 1) {
 8         x = large_mod_mul(x, x, n);
 9         if (x == 1) return false;
10         if (x == n-1) return true;
11     }
12     return false;
13 }
14
15 // test whether n is prime
16 // based on miller-rabin test
17 // O(logn*logn)
18 bool is_prime(ull n) {
19     if (n == 2) return true;
20     if (n < 2 || n % 2 == 0) return false;
21
22     ull d = n >> 1, s = 1;
23     for(; (d&1) == 0; s++) d >>= 1;
24
25 #define T(a) test_witness(a##ull, n, s)
26     if (n < 4759123141ull) return T(2) && T(7) && T(61);
27     return T(2) && T(325) && T(9375) && T(28178)
28         && T(450775) && T(9780504) && T(1795265022);
```

```
29 #undef T
30 }
```

## 2.4   Chinese Remainder Theorem

```
 1 // find x s.t.  x === a[0] (mod n[0])
 2 //              === a[1] (mod n[1])
 3 //              ...
 4 // assumption: gcd(n[i], n[j]) = 1
 5 ll chinese_remainder(ll* a, ll* n, int size) {
 6     if (size == 1) return *a;
 7     ll tmp = modinverse(n[0], n[1]);
 8     ll tmp2 = (tmp * (a[1] - a[0]) % n[1] + n[1]) % n[1];
 9     ll ora = a[1];
10     ll tgcd = gcd(n[0], n[1]);
11     a[1] = a[0] + n[0] / tgcd * tmp2;
12     n[1] *= n[0] / tgcd;
13     ll ret = chinese_remainder(a + 1, n + 1, size - 1);
14     n[1] /= n[0] / tgcd;
15     a[1] = ora;
16     return ret;
17 }
```

## 2.5   Modular Equation

$x \equiv a \pmod{m}$, $x \equiv b \pmod{n}$을 만족시키는 $x$를 구하는 방법.

$m$과 $n$을 소인수분해한 후 소수의 제곱꼴의 합동식들로 각각 쪼갠다. 이 때 특정 소수에 대하여 모순이 생기면 불가능한 경우고, 모든 소수에 대해서 모순이 생기지 않으면 전체 식을 CRT로 합치면 된다. 이제 $x \equiv x_1 \pmod{p^{k_1}}$과 $x \equiv x_2 \pmod{p^{k_2}}$가 모순이 생길 조건은 $k_1 \leq k_2$라고 했을 때, $x_1 \not\equiv x_2 \pmod{p^{k_1}}$인 경우이다. 모순이 생기지 않았을 때 답을 구하려면 CRT로 합칠 때 $x \equiv x_2 \pmod{p^{k_2}}$만을 남기고 합쳐주면 된다.

## 2.6   Rational Number Class

```
 1 struct rational {
 2     long long p, q;
 3
 4     void red() {
 5         if (q < 0) {
 6             p = -p;
 7             q = -q;
 8         }
 9         ll t = gcd((p >= 0 ? p : -p), q);
10         p /= t;
11         q /= t;
12     }
13
14     rational(): p(0), q(1) {}
15     rational(long long p_): p(p_), q(1) {}
```

```cpp
    rational(long long p_, long long q_): p(p_), q(q_) { red(); }

    bool operator==(const rational& rhs) const {
        return p == rhs.p && q == rhs.q;
    }
    bool operator!=(const rational& rhs) const {
        return p != rhs.p || q != rhs.q;
    }
    bool operator<(const rational& rhs) const {
        return p * rhs.q < rhs.p * q;
    }
    rational operator+(const rational& rhs) const {
        ll g = gcd(q, rhs.q);
        return rational(p * (rhs.q / g) + rhs.p * (q / g), (q / g) * rhs.q);
    }
    rational operator-(const rational& rhs) const {
        ll g = gcd(q, rhs.q);
        return rational(p * (rhs.q / g) - rhs.p * (q / g), (q / g) * rhs.q);
    }
    rational operator*(const rational& rhs) const {
        return rational(p * rhs.p, q * rhs.q);
    }
    rational operator/(const rational& rhs) const {
        return rational(p * rhs.q, q * rhs.p);
    }
};
```

## 2.7 Catalan number

다양한 문제의 답이 되는 수열이다.

- 길이가 $2n$인 올바른 괄호 수식의 수
- $n+1$개의 리프를 가진 풀 바이너리 트리의 수
- $n+2$각형을 $n$개의 삼각형으로 나누는 방법의 수

$$C_n = \frac{1}{n+1}\binom{2n}{n}$$

$$C_0 = 1 \quad \text{and} \quad C_{n+1} = \sum_{i=0}^{n} C_i C_{n-i}$$

$$C_0 = 1 \quad \text{and} \quad C_{n+1} = \frac{2(2n+1)}{n+2} C_n$$

## 2.8 Burnside's Lemma

경우의 수를 세는데, 특정 transform operation(회전, 반사, ..)해서 같은 경우들은 하나로 친다. 전체 경우의 수는?

- 각 operation마다 이 operation을 했을 때 변하지 않는 경우의 수를 센다 (단, "아무것도 하지 않는다"라는 operation도 있어야 함!)

- 전체 경우의 수를 더한 후, operation의 수로 나눈다. (답이 맞다면 항상 나누어 떨어져야 한다)

## 2.9 Kirchoff's Theorem

그래프의 스패닝 트리의 개수를 구하는 정리.

무향 그래프의 Laplacian matrix $L$를 만든다. 이것은 (정점의 차수 대각 행렬) - (인접행렬)이다. $L$에서 행과 열을 하나씩 제거한 것을 $L'$라 하자. 어느 행/열이든 관계 없다. 그래프의 스패닝 트리의 개수는 $det(L')$이다.

## 2.10 Lucas Theorem

```cpp
// calculate nCm % p  when p is prime
int lucas_theorem(const char *n, const char *m, int p) {
    vector<int> np, mp;
    int i;
    for (i = 0; n[i]; i++) {
        if (n[i] == '0' && np.empty()) continue;
        np.push_back(n[i] - '0');
    }
    for (i = 0; m[i]; i++) {
        if (m[i] == '0' && mp.empty()) continue;
        mp.push_back(m[i] - '0');
    }

    int ret = 1;
    int ni = 0, mi = 0;
    while (ni < np.size() || mi < mp.size()) {
        int nmod = 0, mmod = 0;
        for (i = ni; i < np.size(); i++) {
            if (i + 1 < np.size())
                np[i + 1] += (np[i] % p) * 10;
            else
                nmod = np[i] % p;
            np[i] /= p;
        }
        for (i = mi; i < mp.size(); i++) {
            if (i + 1 < mp.size())
                mp[i + 1] += (mp[i] % p) * 10;
            else
                mmod = mp[i] % p;
            mp[i] /= p;
        }
        while (ni < np.size() && np[ni] == 0) ni++;
        while (mi < mp.size() && mp[mi] == 0) mi++;
        // implement binomial. binomial(m,n) = 0 if m < n
```

```
35          ret = (ret * binomial(nmod, mmod)) % p;
36      }
37      return ret;
38 }
```

## 2.11   Fast Fourier Transform

```
1  void fft(int sign, int n, double *real, double *imag) {
2      double theta = sign * 2 * pi / n;
3      for (int m = n; m >= 2; m >>= 1, theta *= 2) {
4          double wr = 1, wi = 0, c = cos(theta), s = sin(theta);
5          for (int i = 0, mh = m >> 1; i < mh; ++i) {
6              for (int j = i; j < n; j += m) {
7                  int k = j + mh;
8                  double xr = real[j] - real[k], xi = imag[j] - imag[k];
9                  real[j] += real[k], imag[j] += imag[k];
10                 real[k] = wr * xr - wi * xi, imag[k] = wr * xi + wi * xr;
11             }
12             double _wr = wr * c - wi * s, _wi = wr * s + wi * c;
13             wr = _wr, wi = _wi;
14         }
15     }
16     for (int i = 1, j = 0; i < n; ++i) {
17         for (int k = n >> 1; k > (j ^= k); k >>= 1);
18         if (j < i) swap(real[i], real[j]), swap(imag[i], imag[j]);
19     }
20 }
21 // Compute Poly(a)*Poly(b), write to r; Indexed from 0
22 // O(n*logn)
23 int mult(int *a, int n, int *b, int m, int *r) {
24     const int maxn = 100;
25     static double ra[maxn], rb[maxn], ia[maxn], ib[maxn];
26     int fn = 1;
27     while (fn < n + m) fn <<= 1; // n + m: interested length
28     for (int i = 0; i < n; ++i) ra[i] = a[i], ia[i] = 0;
29     for (int i = n; i < fn; ++i) ra[i] = ia[i] = 0;
30     for (int i = 0; i < m; ++i) rb[i] = b[i], ib[i] = 0;
31     for (int i = m; i < fn; ++i) rb[i] = ib[i] = 0;
32     fft(1, fn, ra, ia);
33     fft(1, fn, rb, ib);
34     for (int i = 0; i < fn; ++i) {
35         double real = ra[i] * rb[i] - ia[i] * ib[i];
36         double imag = ra[i] * ib[i] + rb[i] * ia[i];
37         ra[i] = real, ia[i] = imag;
38     }
39     fft(-1, fn, ra, ia);
40     for (int i = 0; i < fn; ++i) r[i] = (int)floor(ra[i] / fn + 0.5);
41     return fn;
42 }
```

## 2.12   Matrix Operations

```
1  const int MATSZ = 100;
```

```
2
3  inline bool is_zero(double a) { return fabs(a) < 1e-9; }
4
5  // out = A^(-1), returns det(A)
6  // A becomes invalid after call this
7  // O(n^3)
8  double inverse_and_det(int n, double A[][MATSZ], double out[][MATSZ]) {
9      double det = 1;
10     for (int i = 0; i < n; i++) {
11         for (int j = 0; j < n; j++) out[i][j] = 0;
12         out[i][i] = 1;
13     }
14     for (int i = 0; i < n; i++) {
15         if (is_zero(A[i][i])) {
16             double maxv = 0;
17             int maxid = -1;
18             for (int j = i + 1; j < n; j++) {
19                 auto cur = fabs(A[j][i]);
20                 if (maxv < cur) {
21                     maxv = cur;
22                     maxid = j;
23                 }
24             }
25             if (maxid == -1 || is_zero(A[maxid][i])) return 0;
26             for (int k = 0; k < n; k++) {
27                 A[i][k] += A[maxid][k];
28                 out[i][k] += out[maxid][k];
29             }
30         }
31         det *= A[i][i];
32         double coeff = 1.0 / A[i][i];
33         for (int j = 0; j < n; j++) A[i][j] *= coeff;
34         for (int j = 0; j < n; j++) out[i][j] *= coeff;
35         for (int j = 0; j < n; j++) if (j != i) {
36             double mp = A[j][i];
37             for (int k = 0; k < n; k++) A[j][k] -= A[i][k] * mp;
38             for (int k = 0; k < n; k++) out[j][k] -= out[i][k] * mp;
39         }
40     }
41     return det;
42 }
```

## 2.13   Gaussian Elimination

```
1  const double EPS = 1e-10;
2  typedef vector<vector<double>> VVD;
3
4  // Gauss-Jordan elimination with full pivoting.
5  // solving systems of linear equations (AX=B)
6  // INPUT:    a[][] = an n*n matrix
7  //           b[][] = an n*m matrix
8  // OUTPUT:   X     = an n*m matrix (stored in b[][])
9  //           A^{-1} = an n*n matrix (stored in a[][])
10 // O(n^3)
```

```
11  bool gauss_jordan(VVD& a, VVD& b) {
12      const int n = a.size();
13      const int m = b[0].size();
14      vector<int> irow(n), icol(n), ipiv(n);
15
16      for (int i = 0; i < n; i++) {
17          int pj = -1, pk = -1;
18          for (int j = 0; j < n; j++) if (!ipiv[j])
19              for (int k = 0; k < n; k++) if (!ipiv[k])
20                  if (pj == -1 || fabs(a[j][k]) > fabs(a[pj][pk])) { pj = j; pk =
                        k; }
21          if (fabs(a[pj][pk]) < EPS) return false; // matrix is singular
22          ipiv[pk]++;
23          swap(a[pj], a[pk]);
24          swap(b[pj], b[pk]);
25          irow[i] = pj;
26          icol[i] = pk;
27
28          double c = 1.0 / a[pk][pk];
29          a[pk][pk] = 1.0;
30          for (int p = 0; p < n; p++) a[pk][p] *= c;
31          for (int p = 0; p < m; p++) b[pk][p] *= c;
32          for (int p = 0; p < n; p++) if (p != pk) {
33              c = a[p][pk];
34              a[p][pk] = 0;
35              for (int q = 0; q < n; q++) a[p][q] -= a[pk][q] * c;
36              for (int q = 0; q < m; q++) b[p][q] -= b[pk][q] * c;
37          }
38      }
39      for (int p = n - 1; p >= 0; p--) if (irow[p] != icol[p]) {
40          for (int k = 0; k < n; k++) swap(a[k][irow[p]], a[k][icol[p]]);
41      }
42      return true;
43  }
```

## 2.14   Simplex Algorithm

```
1  // Two-phase simplex algorithm for solving linear programs of the form
2  //     maximize     c^T x
3  //     subject to   Ax <= b
4  //                  x >= 0
5  // INPUT: A -- an m x n matrix
6  //        b -- an m-dimensional vector
7  //        c -- an n-dimensional vector
8  //        x -- a vector where the optimal solution will be stored
9  // OUTPUT: value of the optimal solution (infinity if unbounded
10 //         above, nan if infeasible)
11 // To use this code, create an LPSolver object with A, b, and c as
12 // arguments.  Then, call Solve(x).
13 typedef vector<double> VD;
14 typedef vector<VD> VVD;
15 typedef vector<int> VI;
16 const double EPS = 1e-9;
17
```

```
18  struct LPSolver {
19      int m, n;
20      VI B, N;
21      VVD D;
22
23      LPSolver(const VVD& A, const VD& b, const VD& c) :
24          m(b.size()), n(c.size()), N(n + 1), B(m), D(m + 2, VD(n + 2)) {
25          for (int i = 0; i < m; i++) for (int j = 0; j < n; j++) D[i][j] = A[i][j
                ];
26          for (int i = 0; i < m; i++) { B[i] = n + i; D[i][n] = -1; D[i][n + 1] =
                b[i]; }
27          for (int j = 0; j < n; j++) { N[j] = j; D[m][j] = -c[j]; }
28          N[n] = -1; D[m + 1][n] = 1;
29      }
30
31      void pivot(int r, int s) {
32          double inv = 1.0 / D[r][s];
33          for (int i = 0; i < m + 2; i++) if (i != r)
34              for (int j = 0; j < n + 2; j++) if (j != s)
35                  D[i][j] -= D[r][j] * D[i][s] * inv;
36          for (int j = 0; j < n + 2; j++) if (j != s) D[r][j] *= inv;
37          for (int i = 0; i < m + 2; i++) if (i != r) D[i][s] *= -inv;
38          D[r][s] = inv;
39          swap(B[r], N[s]);
40      }
41
42      bool simplex(int phase) {
43          int x = phase == 1 ? m + 1 : m;
44          while (true) {
45              int s = -1;
46              for (int j = 0; j <= n; j++) {
47                  if (phase == 2 && N[j] == -1) continue;
48                  if (s == -1 || D[x][j] < D[x][s] || D[x][j] == D[x][s] && N[j] <
                        N[s]) s = j;
49              }
50              if (D[x][s] > -EPS) return true;
51              int r = -1;
52              for (int i = 0; i < m; i++) {
53                  if (D[i][s] < EPS) continue;
54                  if (r == -1 || D[i][n + 1] / D[i][s] < D[r][n + 1] / D[r][s] ||
55                      (D[i][n + 1] / D[i][s]) == (D[r][n + 1] / D[r][s]) && B[i] <
                            B[r]) r = i;
56              }
57              if (r == -1) return false;
58              pivot(r, s);
59          }
60      }
61
62      double solve(VD& x) {
63          int r = 0;
64          for (int i = 1; i < m; i++) if (D[i][n + 1] < D[r][n + 1]) r = i;
65          if (D[r][n + 1] < -EPS) {
66              pivot(r, n);
67              if (!simplex(1) || D[m + 1][n + 1] < -EPS)
68                  return -numeric_limits<double>::infinity();
```

```
69          for (int i = 0; i < m; i++) if (B[i] == -1) {
70              int s = -1;
71              for (int j = 0; j <= n; j++)
72                  if (s == -1 || D[i][j] < D[i][s] || D[i][j] == D[i][s] && N[
                        j] < N[s]) s = j;
73              pivot(i, s);
74          }
75      }
76      if (!simplex(2))
77          return numeric_limits<double>::infinity();
78      x = VD(n);
79      for (int i = 0; i < m; i++) if (B[i] < n) x[B[i]] = D[i][n + 1];
80      return D[m][n + 1];
81  }
82 };
```

# 3 Data Structure

## 3.1 Order statistic tree

```
1  #include <ext/pb_ds/assoc_container.hpp>
2  #include <ext/pb_ds/tree_policy.hpp>
3  #include <ext/pb_ds/detail/standard_policies.hpp>
4  #include <functional>
5  #include <iostream>
6  using namespace __gnu_pbds;
7  using namespace std;
8
9  // tree<key_type, value_type(set if null), comparator, ...>
10 using ordered_set = tree<int, null_type, less<int>, rb_tree_tag,
11     tree_order_statistics_node_update>;
12
13 int main()
14 {
15     ordered_set X;
16     for (int i = 1; i < 10; i += 2) X.insert(i); // 1 3 5 7 9
17     cout << boolalpha;
18     cout << *X.find_by_order(2) << endl; // 5
19     cout << *X.find_by_order(4) << endl; // 9
20     cout << (X.end() == X.find_by_order(5)) << endl; // true
21
22     cout << X.order_of_key(-1) << endl; // 0
23     cout << X.order_of_key(1) << endl; // 0
24     cout << X.order_of_key(4) << endl; // 2
25     X.erase(3);
26     cout << X.order_of_key(4) << endl; // 1
27     for (int t : X) printf("%d␣", t); // 1 5 7 9
28 }
```

## 3.2 Fenwick Tree

```
1  const int TSIZE = 100000;
```

```
2  int tree[TSIZE + 1];
3
4  // Returns the sum from index 1 to p, inclusive
5  int query(int p) {
6      int ret = 0;
7      for (; p > 0; p -= p & -p) ret += tree[p];
8      return ret;
9  }
10
11 // Adds val to element with index pos
12 void add(int p, int val) {
13     for (; p <= TSIZE; p += p & -p) tree[p] += val;
14 }
```

## 3.3 Segment Tree with Lazy Propagation

```
1  // example implementation of sum tree
2  const int TSIZE = 131072; // always 2^k form && n <= TSIZE
3  int segtree[TSIZE * 2], prop[TSIZE * 2];
4  void seg_init(int nod, int l, int r) {
5      if (l == r) segtree[nod] = dat[l];
6      else {
7          int m = (l + r) >> 1;
8          seg_init(nod << 1, l, m);
9          seg_init(nod << 1 | 1, m + 1, r);
10         segtree[nod] = segtree[nod << 1] + segtree[nod << 1 | 1];
11     }
12 }
13 void seg_relax(int nod, int l, int r) {
14     if (prop[nod] == 0) return;
15     if (l < r) {
16         int m = (l + r) >> 1;
17         segtree[nod << 1] += (m - l + 1) * prop[nod];
18         prop[nod << 1] += prop[nod];
19         segtree[nod << 1 | 1] += (r - m) * prop[nod];
20         prop[nod << 1 | 1] += prop[nod];
21     }
22     prop[nod] = 0;
23 }
24 int seg_query(int nod, int l, int r, int s, int e) {
25     if (r < s || e < l) return 0;
26     if (s <= l && r <= e) return segtree[nod];
27     seg_relax(nod, l, r);
28     int m = (l + r) >> 1;
29     return seg_query(nod << 1, l, m, s, e) + seg_query(nod << 1 | 1, m + 1, r, s
         , e);
30 }
31 void seg_update(int nod, int l, int r, int s, int e, int val) {
32     if (r < s || e < l) return;
33     if (s <= l && r <= e) {
34         segtree[nod] += (r - l + 1) * val;
35         prop[nod] += val;
36         return;
37     }
```

```
38        seg_relax(nod, l, r);
39        int m = (l + r) >> 1;
40        seg_update(nod << 1, l, m, s, e, val);
41        seg_update(nod << 1 | 1, m + 1, r, s, e, val);
42        segtree[nod] = segtree[nod << 1] + segtree[nod << 1 | 1];
43 }
44 // usage:
45 // seg_update(1, 0, n - 1, qs, qe, val);
46 // seg_query(1, 0, n - 1, qs, qe);
```

## 3.4  Persistent Segment Tree

```
1  // persistent segment tree impl: sum tree
2  namespace pstree {
3      typedef int val_t;
4      const int DEPTH = 18;
5      const int TSIZE = 1 << 18;
6      const int MAX_QUERY = 262144;
7
8      struct node {
9          val_t v;
10         node *l, *r;
11     } npoll[TSIZE * 2 + MAX_QUERY * (DEPTH + 1)];
12
13     int pptr, last_q;
14
15     node *head[MAX_QUERY + 1];
16     int q[MAX_QUERY + 1];
17     int lqidx;
18
19     void init() {
20         // zero-initialize, can be changed freely
21         memset(&npoll[TSIZE - 1], 0, sizeof(node) * TSIZE);
22
23         for (int i = TSIZE - 2; i >= 0; i--) {
24             npoll[i].v = 0;
25             npoll[i].l = &npoll[i*2+1];
26             npoll[i].r = &npoll[i*2+2];
27         }
28
29         head[0] = &npoll[0];
30         last_q = 0;
31         pptr = 2 * TSIZE - 1;
32         q[0] = 0;
33         lqidx = 0;
34     }
35
36     // update val to pos at time t
37     // 0 <= t <= MAX_QUERY, 0 <= pos < TSIZE
38     void update(int pos, int val, int t, int prev) {
39         head[++last_q] = &npoll[pptr++];
40         node *old = head[q[prev]], *now = head[last_q];
41         while (lqidx < t) q[lqidx++] = q[prev];
42         q[t] = last_q;
```

```
43
44         int flag = 1 << DEPTH;
45         for (;;) {
46             now->v = old->v + val;
47             flag >>= 1;
48             if (flag==0) {
49                 now->l = now->r = nullptr; break;
50             }
51             if (flag & pos) {
52                 now->l = old->l;
53                 now->r = &npoll[pptr++];
54                 now = now->r, old = old->r;
55             } else {
56                 now->r = old->r;
57                 now->l = &npoll[pptr++];
58                 now = now->l, old = old->l;
59             }
60         }
61     }
62
63     val_t query(int s, int e, int l, int r, node *n) {
64         if (s == l && e == r) return n->v;
65         int m = (l + r) / 2;
66         if (m >= e) return query(s, e, l, m, n->l);
67         else if (m < s) return query(s, e, m + 1, r, n->r);
68         else return query(s, m, l, m, n->l) + query(m + 1, e, m + 1, r, n->r);
69     }
70
71     // query summation of [s, e] at time t
72     val_t query(int s, int e, int t) {
73         s = max(0, s); e = min(TSIZE - 1, e);
74         if (s > e) return 0;
75         return query(s, e, 0, TSIZE - 1, head[q[t]]);
76     }
77 }
```

## 3.5  Splay Tree

```
1  // example : https://www.acmicpc.net/problem/13159
2  struct node {
3      node* l, * r, * p;
4      int cnt, min, max, val;
5      long long sum;
6      bool inv;
7      node(int _val) :
8          cnt(1), sum(_val), min(_val), max(_val), val(_val), inv(false),
9          l(nullptr), r(nullptr), p(nullptr) {
10     }
11 };
12 node* root;
13
14 void update(node* x) {
15     x->cnt = 1;
16     x->sum = x->min = x->max = x->val;
```

```cpp
17      if (x->l) {
18          x->cnt += x->l->cnt;
19          x->sum += x->l->sum;
20          x->min = min(x->min, x->l->min);
21          x->max = max(x->max, x->l->max);
22      }
23      if (x->r) {
24          x->cnt += x->r->cnt;
25          x->sum += x->r->sum;
26          x->min = min(x->min, x->r->min);
27          x->max = max(x->max, x->r->max);
28      }
29  }
30
31  void rotate(node* x) {
32      node* p = x->p;
33      node* b = nullptr;
34      if (x == p->l) {
35          p->l = b = x->r;
36          x->r = p;
37      }
38      else {
39          p->r = b = x->l;
40          x->l = p;
41      }
42      x->p = p->p;
43      p->p = x;
44      if (b) b->p = p;
45      x->p ? (p == x->p->l ? x->p->l : x->p->r) = x : (root = x);
46      update(p);
47      update(x);
48  }
49
50  // make x into root
51  void splay(node* x) {
52      while (x->p) {
53          node* p = x->p;
54          node* g = p->p;
55          if (g) rotate((x == p->l) == (p == g->l) ? p : x);
56          rotate(x);
57      }
58  }
59
60  void relax_lazy(node* x) {
61      if (!x->inv) return;
62      swap(x->l, x->r);
63      x->inv = false;
64      if (x->l) x->l->inv = !x->l->inv;
65      if (x->r) x->r->inv = !x->r->inv;
66  }
67
68  // find kth node in splay tree
69  void find_kth(int k) {
70      node* x = root;
71      relax_lazy(x);
72      while (true) {
73          while (x->l && x->l->cnt > k) {
74              x = x->l;
75              relax_lazy(x);
76          }
77          if (x->l) k -= x->l->cnt;
78          if (!k--) break;
79          x = x->r;
80          relax_lazy(x);
81      }
82      splay(x);
83  }
84
85  // collect [l, r] nodes into one subtree and return its root
86  node* interval(int l, int r) {
87      find_kth(l - 1);
88      node* x = root;
89      root = x->r;
90      root->p = nullptr;
91      find_kth(r - l + 1);
92      x->r = root;
93      root->p = x;
94      root = x;
95      return root->r->l;
96  }
97
98  void traverse(node* x) {
99      relax_lazy(x);
100     if (x->l) {
101         traverse(x->l);
102     }
103     // do something
104     if (x->r) {
105         traverse(x->r);
106     }
107 }
108
109 void uptree(node* x) {
110     if (x->p) {
111         uptree(x->p);
112     }
113     relax_lazy(x);
114 }
```

### 3.6 Link/Cut Tree

# 4 DP

## 4.1 Convex Hull Optimization

### 4.1.1 requirement

$O(n^2) \rightarrow O(n \log n)$

조건 1) DP 점화식 꼴

$D[i] = \min_{j<i}(D[j] + b[j] * a[i])$

조건 2) $b[j] \leq b[j+1]$

특수조건) $a[i] \leq a[i+1]$ 도 만족하는 경우, 마지막 쿼리의 위치를 저장해두면 이분검색이 필요없어지기 때문에 amortized $O(n)$ 에 해결할 수 있음

### 4.1.2 Source Code

```
1  //O(n^3) -> O(n^2)
2
3  #define sz 100001
4  long long s[sz];
5  long long dp[2][sz];
6  //deque {index, x pos }
7  int dqi[sz];
8  long long dqm[sz];
9  //pointer to deque
10 int ql,qr;
11 //dp[i][j] = max(dp[i][k] + s[j]*s[k] - s[k]^2)
12 //let y = dp[i][j], x = s[j] -> y = max(s[k]*x + dp[i][k] - s[k]^2);
13
14 //push new value to deque
15 //i = index, x = current x pos
16 void setq(int i, int x)
17 {
18     //a1,b1 = prv line, a2,b2 = new line
19     int a1, a2 = s[i];
20     long long b1, b2 = dp[0][i] - s[i] * s[i], r;
21     //renew deque
22     while (qr>=ql)
23     {
24         //last line enqueued
25         a1 = s[dqi[qr]];
26         b1 = dp[0][dqi[qr]] - s[dqi[qr]] * s[dqi[qr]];
27         //tie breaking to newer one
28         if (a1 == a2)
29         {
30             dqi[qr] = i;
31             return;
32         }
33         // x intersection between last line and new line
34         r = (b1 - b2) / (a2 - a1);
35         if ((b1 - b2) % (a2 - a1)) r++;
36         //last line is not needed
37         if (r <= dqm[qr])
38         {
39             qr--;
40         }
41         else break;
42     }
43     if (r < 0) r = 0;
44     //push back new line
45     if (dqm[qr] < s[n - 1] && r <= s[n - 1])
46     {
47         dqi[++qr] = i;
48         dqm[qr] = r;
49     }
50     //discard old lines
51     while (qr-ql && dqm[ql+1] <= x)
52     {
53         ql++;
54     }
55 }
56
57 int main()
58 {
59     for (int j = 0; j < k; j++)
60     {
61         ql = 0;
62         qr = 1;
63         dqi[0] = dqm[0] = 0;
64         for (int i = 1; i < n; i++)
65         {
66             //get line used by current x pos
67             setq(i, s[i]);
68             //line index to use
69             int g = dqi[ql];
70             //set dp value
71             dp[1][i] = dp[0][g] + s[g] * (s[i] - s[g]);
72         }
73         for (int i = 0; i < n; i++)
74         {
75             dp[0][i] = dp[1][i];
76             dp[1][i] = 0;
77         }
78     }
79 }
```

## 4.2 Divide & Conquer Optimization

$O(kn^2) \rightarrow O(kn \log n)$

조건 1) DP 점화식 꼴

$D[t][i] = \min_{j<i}(D[t-1][j] + C[j][i])$

조건 2) $A[t][i]$ 는 $D[t][i]$ 의 답이 되는 최소의 $j$ 라 할 때, 아래의 부등식을 만족해야 함

$A[t][i] \leq A[t][i+1]$

조건 2-1) 비용 $C$ 가 다음의 사각부등식을 만족하는 경우도 조건 2)를 만족하게 됨

$C[a][c] + C[b][d] \leq C[a][d] + C[b][c] \;\; (a \leq b \leq c \leq d)$

## 4.3  Knuth Optimization

$O(n^3) \rightarrow O(n^2)$

조건 1) DP 점화식 꼴

$D[i][j] = \min_{i<k<j}(D[i][k] + D[k][j]) + C[i][j]$

조건 2) 사각 부등식

$C[a][c] + C[b][d] \leq C[a][d] + C[b][c] \;\; (a \leq b \leq c \leq d)$

조건 3) 단조성

$C[b][c] \leq C[a][d] \;\; (a \leq b \leq c \leq d)$

결론) 조건 2, 3을 만족한다면 $A[i][j]$ 를 $D[i][j]$ 의 답이 되는 최소의 $k$ 라 할 때, 아래의 부등식을 만족하게 됨

$A[i][j-1] \leq A[i][j] \leq A[i+1][j]$

3중 루프를 돌릴 때 위 조건을 이용하면 최종적으로 시간복잡도가 $O(n^2)$ 이 됨

# 5  Graph

## 5.1  SCC (Tarjan)

```
1  const int MAXN = 100;
2  vector<int> graph[MAXN];
3  int up[MAXN], visit[MAXN], vtime;
4  vector<int> stk;
5  int scc_idx[MAXN], scc_cnt;
6
7  void dfs(int nod) {
8      up[nod] = visit[nod] = ++vtime;
9      stk.push_back(nod);
10     for (int next : graph[nod]) {
11         if (visit[next] == 0) {
12             dfs(next);
13             up[nod] = min(up[nod], up[next]);
```

```
14         }
15         else if (scc_idx[next] == 0)
16             up[nod] = min(up[nod], visit[next]);
17     }
18     if (up[nod] == visit[nod]) {
19         ++scc_cnt;
20         int t;
21         do {
22             t = stk.back();
23             stk.pop_back();
24             scc_idx[t] = scc_cnt;
25         } while (!stk.empty() && t != nod);
26     }
27 }
28
29 // find SCCs in given directed graph
30 // O(V+E)
31 void get_scc() {
32     vtime = 0;
33     memset(visit, 0, sizeof(visit));
34     scc_cnt = 0;
35     memset(scc_idx, 0, sizeof(scc_idx));
36     for (int i = 0; i < n; ++i)
37         if (visit[i] == 0) dfs(i);
38 }
```

## 5.2  SCC (Kosaraju)

```
1  const int MAXN = 100;
2  vector<int> graph[MAXN], grev[MAXN];
3  int visit[MAXN], vcnt;
4  int scc_idx[MAXN], scc_cnt;
5  vector<int> emit;
6
7  void dfs(int nod, vector<int> graph[]) {
8      visit[nod] = vcnt;
9      for (int next : graph[nod]) {
10         if (visit[next] == vcnt) continue;
11         dfs(next, graph);
12     }
13     emit.push_back(nod);
14 }
15
16 // find SCCs in given graph
17 // O(V+E)
18 void get_scc() {
19     scc_cnt = 0;
20     vcnt = 1;
21     emit.clear();
22     memset(visit, 0, sizeof(visit));
23
24     for (int i = 0; i < n; i++) {
25         if (visit[i] == vcnt) continue;
26         dfs(i, graph);
```

```
27        }
28    ++vcnt;
29    for (auto st : vector<int>(emit.rbegin(), emit.rend())) {
30        if (visit[st] == vcnt) continue;
31        emit.clear();
32        dfs(st, grev);
33        ++scc_cnt;
34        for (auto node : emit)
35            scc_idx[node] = scc_cnt;
36    }
37 }
38
```

## 5.3 2-SAT

$(b_x \vee b_y) \wedge (\neg b_x \vee b_z) \wedge (b_z \vee \neg b_x) \wedge \cdots$ 같은 form을 2-CNF라고 함. 주어진 2-CNF 식을 참으로 하는 $\{b_1, b_2, \cdots\}$ 가 존재하는지, 존재한다면 그 값은 무엇인지 구하는 문제를 2-SAT 이라 함.

boolean variable $b_i$ 마다 $b_i$ 를 나타내는 정점, $\neg b_i$ 를 나타내는 정점 2개를 만듦. 각 clause $b_i \vee b_j$ 마다 $\neg b_i \rightarrow b_j, \neg b_j \rightarrow b_i$ 이렇게 edge를 이어줌. 그렇게 만든 그래프에서 SCC를 다 구함. 어떤 SCC 안에 $b_i$ 와 $\neg b_i$ 가 같이 포함되어있다면 해가 존재하지 않음. 아니라면 해가 존재함.

해가 존재할 때 구체적인 해를 구하는 방법. 위에서 SCC를 구하면서 SCC DAG를 만들어 준다. 거기서 위상정렬을 한 후, 앞에서부터 SCC를 하나씩 봐준다. 현재 보고있는 SCC에 $b_i$ 가 속해있는데 얘가 $\neg b_i$ 보다 먼저 등장했다면 $b_i = $ false, 반대의 경우라면 $b_i = $ true, 이미 값이 assign되었다면 pass.

## 5.4 BCC, Cut vertex, Bridge

```
1 const int MAXN = 100;
2 vector<pair<int, int>> graph[MAXN];  // { next vertex id, edge id }
3 int up[MAXN], visit[MAXN], vtime;
4 vector<pair<int, int>> stk;
5
6 int is_cut[MAXN];              // v is cut vertex if  is_cut[v] > 0
7 vector<int> bridge;            // list of edge ids
8 vector<int> bcc_idx[MAXN];     // list of bccids for vertex i
9 int bcc_cnt;
10
11 void dfs(int nod, int par_edge) {
12     up[nod] = visit[nod] = ++vtime;
13     int child = 0;
14     for (const auto& e : graph[nod]) {
15         int next = e.first, edge_id = e.second;
16         if (edge_id == par_edge) continue;
17         if (visit[next] == 0) {
18             stk.push_back({ nod, next });
19             ++child;
20             dfs(next, edge_id);
21             if (up[next] == visit[next]) bridge.push_back(edge_id);
22             if (up[next] >= visit[nod]) {
23                 ++bcc_cnt;
24                 do {
25                     auto last = stk.back();
26                     stk.pop_back();
27                     bcc_idx[last.second].push_back(bcc_cnt);
28                     if (last == pair<int, int>{ nod, next }) break;
29                 } while (!stk.empty());
30                 bcc_idx[nod].push_back(bcc_cnt);
31                 is_cut[nod]++;
32             }
33             up[nod] = min(up[nod], up[next]);
34         }
35         else
36             up[nod] = min(up[nod], visit[next]);
37     }
38     if (par_edge == -1 && is_cut[nod] == 1)
39         is_cut[nod] = 0;
40 }
41
42 // find BCCs & cut vertexs & bridges in undirected graph
43 // O(V+E)
44 void get_bcc() {
45     vtime = 0;
46     memset(visit, 0, sizeof(visit));
47     memset(is_cut, 0, sizeof(is_cut));
48     bridge.clear();
49     for (int i = 0; i < n; ++i) bcc_idx[i].clear();
50     bcc_cnt = 0;
51     for (int i = 0; i < n; ++i) {
52         if (visit[i] == 0)
53             dfs(i, -1);
54     }
55 }
```

## 5.5 Shortest Path Faster Algorithm

```
1 // shortest path faster algorithm
2 // average for random graph : O(E) , worst : O(VE)
3
4 const int MAXN = 20001;
5 const int INF = 100000000;
6 int n, m;
7 vector<pair<int, int>> graph[MAXN];
8 bool inqueue[MAXN];
9 int dist[MAXN];
10
11 void spfa(int st) {
12     for (int i = 0; i < n; ++i) {
13         dist[i] = INF;
14     }
15     dist[st] = 0;
```

```
16
17    queue<int> q;
18    q.push(st);
19    inqueue[st] = true;
20    while (!q.empty()) {
21        int u = q.front();
22        q.pop();
23        inqueue[u] = false;
24        for (auto& e : graph[u]) {
25            if (dist[u] + e.second < dist[e.first]) {
26                dist[e.first] = dist[u] + e.second;
27                if (!inqueue[e.first]) {
28                    q.push(e.first);
29                    inqueue[e.first] = true;
30                }
31            }
32        }
33    }
34 }
```

## 5.6  Lowest Common Ancestor

```
1 const int MAXN = 100;
2 const int MAXLN = 9;
3 vector<int> tree[MAXN];
4 int depth[MAXN];
5 int par[MAXLN][MAXN];
6
7 void dfs(int nod, int parent) {
8     for (int next : tree[nod]) {
9         if (next == parent) continue;
10        depth[next] = depth[nod] + 1;
11        par[0][next] = nod;
12        dfs(next, nod);
13    }
14 }
15
16 void prepare_lca() {
17    const int root = 0;
18    dfs(root, -1);
19    par[0][root] = root;
20    for (int i = 1; i < MAXLN; ++i)
21        for (int j = 0; j < n; ++j)
22            par[i][j] = par[i - 1][par[i - 1][j]];
23 }
24
25 // find lowest common ancestor in tree between u & v
26 // assumption : must call 'prepare_lca' once before call this
27 // O(logV)
28 int lca(int u, int v) {
29    if (depth[u] < depth[v]) swap(u, v);
30    if (depth[u] > depth[v]) {
31        for (int i = MAXLN - 1; i >= 0; --i)
32            if (depth[u] - (1 << i) >= depth[v])
```

```
33                u = par[i][u];
34        }
35        if (u == v) return u;
36        for (int i = MAXLN - 1; i >= 0; --i) {
37            if (par[i][u] != par[i][v]) {
38                u = par[i][u];
39                v = par[i][v];
40            }
41        }
42        return par[0][u];
43 }
```

## 5.7  Heavy-Light Decomposition

```
1 // heavy-light decomposition
2 //
3 // hld h;
4 // insert edges to tree[0~n-1];
5 // h.init(n);
6 // h.decompose(root);
7 // h.hldquery(u, v); // edges from u to v
8 struct hld {
9     static const int MAXLN = 18;
10    static const int MAXN = 1 << (MAXLN - 1);
11    vector<int> tree[MAXN];
12    int subsize[MAXN], depth[MAXN], pa[MAXLN][MAXN];
13
14    int chead[MAXN], cidx[MAXN];
15    int lchain;
16    int flatpos[MAXN + 1], fptr;
17
18    void dfs(int u, int par) {
19        pa[0][u] = par;
20        subsize[u] = 1;
21        for (int v : tree[u]) {
22            if (v == pa[0][u]) continue;
23            depth[v] = depth[u] + 1;
24            dfs(v, u);
25            subsize[u] += subsize[v];
26        }
27    }
28
29    void init(int size)
30    {
31        lchain = fptr = 0;
32        dfs(0, -1);
33        memset(chead, -1, sizeof(chead));
34
35        for (int i = 1; i < MAXLN; i++) {
36            for (int j = 0; j < size; j++) {
37                if (pa[i - 1][j] != -1) {
38                    pa[i][j] = pa[i - 1][pa[i - 1][j]];
39                }
40            }
```

```
 41             }
 42         }
 43
 44     void decompose(int u) {
 45         if (chead[lchain] == -1) chead[lchain] = u;
 46         cidx[u] = lchain;
 47         flatpos[u] = ++fptr;
 48
 49         int maxchd = -1;
 50         for (int v : tree[u]) {
 51             if (v == pa[0][u]) continue;
 52             if (maxchd == -1 || subsize[maxchd] < subsize[v]) maxchd = v;
 53         }
 54         if (maxchd != -1) decompose(maxchd);
 55
 56         for (int v : tree[u]) {
 57             if (v == pa[0][u] || v == maxchd) continue;
 58             ++lchain; decompose(v);
 59         }
 60     }
 61
 62     int lca(int u, int v) {
 63         if (depth[u] < depth[v]) swap(u, v);
 64
 65         int logu;
 66         for (logu = 1; 1 << logu <= depth[u]; logu++);
 67         logu--;
 68
 69         int diff = depth[u] - depth[v];
 70         for (int i = logu; i >= 0; --i) {
 71             if ((diff >> i) & 1) u = pa[i][u];
 72         }
 73         if (u == v) return u;
 74
 75         for (int i = logu; i >= 0; --i) {
 76             if (pa[i][u] != pa[i][v]) {
 77                 u = pa[i][u];
 78                 v = pa[i][v];
 79             }
 80         }
 81         return pa[0][u];
 82     }
 83
 84     // TODO: implement query functions
 85     inline int query(int s, int e) {
 86         return 0;
 87     }
 88
 89     int subquery(int u, int v, int t) {
 90         int uchain, vchain = cidx[v];
 91         int ret = 0;
 92         for (;;) {
 93             uchain = cidx[u];
 94             if (uchain == vchain) {
 95                 ret += query(flatpos[v], flatpos[u]);
 96                 break;
 97             }
 98
 99             ret += query(flatpos[chead[uchain]], flatpos[u]);
100             u = pa[0][chead[uchain]];
101         }
102         return ret;
103     }
104
105     inline int hldquery(int u, int v) {
106         int p = lca(u, v);
107         return subquery(u, p) + subquery(v, p) - query(flatpos[p], flatpos[p]);
108     }
109 };
```

## 5.8 Bipartite Matching (Hopcroft-Karp)

```
 1 // in: n, m, graph
 2 // out: match, matched
 3 // vertex cover: (reached[0][left_node] == 0) || (reached[1][right_node] == 1)
 4 // O(E*sqrt(V))
 5 struct BipartiteMatching {
 6     int n, m;
 7     vector<vector<int>> graph;
 8     vector<int> matched, match, edgeview, level;
 9     vector<int> reached[2];
10     BipartiteMatching(int n, int m) : n(n), m(m), graph(n), matched(m, -1),
      match(n, -1) {}
11
12     bool assignLevel() {
13         bool reachable = false;
14         level.assign(n, -1);
15         reached[0].assign(n, 0);
16         reached[1].assign(m, 0);
17         queue<int> q;
18         for (int i = 0; i < n; i++) {
19             if (match[i] == -1) {
20                 level[i] = 0;
21                 reached[0][i] = 1;
22                 q.push(i);
23             }
24         }
25         while (!q.empty()) {
26             auto cur = q.front(); q.pop();
27             for (auto adj : graph[cur]) {
28                 reached[1][adj] = 1;
29                 auto next = matched[adj];
30                 if (next == -1) {
31                     reachable = true;
32                 }
33                 else if (level[next] == -1) {
34                     level[next] = level[cur] + 1;
35                     reached[0][next] = 1;
36                     q.push(next);
```

```
37                    }
38                }
39            }
40            return reachable;
41        }
42
43    int findpath(int nod) {
44        for (int &i = edgeview[nod]; i < graph[nod].size(); i++) {
45            int adj = graph[nod][i];
46            int next = matched[adj];
47            if (next >= 0 && level[next] != level[nod] + 1) continue;
48            if (next == -1 || findpath(next)) {
49                match[nod] = adj;
50                matched[adj] = nod;
51                return 1;
52            }
53        }
54        return 0;
55    }
56
57    int solve() {
58        int ans = 0;
59        while (assignLevel()) {
60            edgeview.assign(n, 0);
61            for (int i = 0; i < n; i++)
62                if (match[i] == -1)
63                    ans += findpath(i);
64        }
65        return ans;
66    }
67 };
```

## 5.9   Maximum Flow (Dinic)

```
 1 // usage:
 2 // MaxFlowDinic::init(n);
 3 // MaxFlowDinic::add_edge(0, 1, 100, 100); // for bidirectional edge
 4 // MaxFlowDinic::add_edge(1, 2, 100); // directional edge
 5 // result = MaxFlowDinic::solve(0, 2); // source -> sink
 6 // graph[i][edgeIndex].res -> residual
 7 //
 8 // in order to find out the minimum cut, use `l`.
 9 // if l[i] == 0, i is unrechable.
10 //
11 // O(V*V*E)
12 // with unit capacities, O(min(V^(2/3), E^(1/2)) * E)
13 struct MaxFlowDinic {
14     typedef int flow_t;
15     struct Edge {
16         int next;
17         int inv; /* inverse edge index */
18         flow_t res; /* residual */
19     };
20     int n;
21    vector<vector<Edge>> graph;
22    vector<int> q, l, start;
23
24    void init(int _n) {
25        n = _n;
26        graph.resize(n);
27        for (int i = 0; i < n; i++) graph[i].clear();
28    }
29    void add_edge(int s, int e, flow_t cap, flow_t caprev = 0) {
30        Edge forward{ e, graph[e].size(), cap };
31        Edge reverse{ s, graph[s].size(), caprev };
32        graph[s].push_back(forward);
33        graph[e].push_back(reverse);
34    }
35    bool assign_level(int source, int sink) {
36        int t = 0;
37        memset(&l[0], 0, sizeof(l[0]) * l.size());
38        l[source] = 1;
39        q[t++] = source;
40        for (int h = 0; h < t && !l[sink]; h++) {
41            int cur = q[h];
42            for (const auto& e : graph[cur]) {
43                if (l[e.next] || e.res == 0) continue;
44                l[e.next] = l[cur] + 1;
45                q[t++] = e.next;
46            }
47        }
48        return l[sink] != 0;
49    }
50    flow_t block_flow(int cur, int sink, flow_t current) {
51        if (cur == sink) return current;
52        for (int& i = start[cur]; i < graph[cur].size(); i++) {
53            auto& e = graph[cur][i];
54            if (e.res == 0 || l[e.next] != l[cur] + 1) continue;
55            if (flow_t res = block_flow(e.next, sink, min(e.res, current))) {
56                e.res -= res;
57                graph[e.next][e.inv].res += res;
58                return res;
59            }
60        }
61        return 0;
62    }
63    flow_t solve(int source, int sink) {
64        q.resize(n);
65        l.resize(n);
66        start.resize(n);
67        flow_t ans = 0;
68        while (assign_level(source, sink)) {
69            memset(&start[0], 0, sizeof(start[0]) * n);
70            while (flow_t flow = block_flow(source, sink, numeric_limits<flow_t
                 >::max()))
71                ans += flow;
72        }
73        return ans;
74    }
```

```
75 };
```

## 5.10   Maximum Flow with Edge Demands

그래프 $G = (V, E)$ 가 있고 source $s$ 와 sink $t$ 가 있다. 각 간선마다 $d(e) \le f(e) \le c(e)$ 를 만족하도록 flow $f(e)$ 를 흘려야 한다. 이 때의 maximum flow를 구하는 문제다.

먼저 모든 demand를 합한 값 $D$ 를 아래와 같이 정의한다.

$$D = \sum_{(u \to v) \in E} d(u \to v)$$

이제 $G$ 에 몇개의 정점과 간선을 추가하여 새로운 그래프 $G' = (V', E')$ 을 만들 것이다. 먼저 새로운 source $s'$ 과 새로운 sink $t'$ 을 추가한다. 그리고 $s'$ 에서 $V$ 의 모든 점마다 간선을 이어주고, $V$ 의 모든 점에서 $t'$ 로 간선을 이어준다.

새로운 capacity function $c'$ 을 아래와 같이 정의한다.

1. $V$ 의 점 $v$ 에 대해 $c'(s' \to v) = \sum_{u \in V} d(u \to v)$ , $c'(v \to t') = \sum_{w \in V} d(v \to w)$

2. $E$ 의 간선 $u \to v$ 에 대해 $c'(u \to v) = c(u \to v) - d(u \to v)$

3. $c'(t \to s) = \infty$

이렇게 만든 새로운 그래프 $G'$ 에서 maximum flow를 구했을 때 그 값이 $D$ 라면 원래 문제의 해가 존재하고, 그 값이 $D$ 가 아니라면 원래 문제의 해는 존재하지 않는다.

위에서 maximum flow를 구하고 난 상태의 residual graph 에서 $s'$ 과 $t'$ 을 떼버리고 $s$ 에서 $t$ 사이의 augment path 를 계속 찾으면 원래 문제의 해를 구할 수 있다.

## 5.11   Min-cost Maximum Flow

```
1  // precondition: there is no negative cycle.
2  // usage:
3  //  MinCostFlow mcf(n);
4  //  for(each edges) mcf.addEdge(from, to, cost, capacity);
5  //  mcf.solve(source, sink); // min cost max flow
6  //  mcf.solve(source, sink, 0); // min cost flow
7  //  mcf.solve(source, sink, goal_flow); // min cost flow with total_flow >=
     goal_flow if possible
8  struct MinCostFlow
9  {
10     typedef int cap_t;
11     typedef int cost_t;
12
13     bool iszerocap(cap_t cap) { return cap == 0; }
14
15     struct edge {
16         int target;
17         cost_t cost;
```

```
18         cap_t residual_capacity;
19         cap_t orig_capacity;
20         size_t revid;
21     };
22
23     int n;
24     vector<vector<edge>> graph;
25     vector<cost_t> pi;
26     bool needNormalize, ranbefore;
27     int lastStart;
28
29     MinCostFlow(int n) : graph(n), n(n), pi(n, 0), needNormalize(false),
         ranbefore(false) {}
30     void addEdge(int s, int e, cost_t cost, cap_t cap)
31     {
32         if (s == e) return;
33         edge forward={e, cost, cap, cap, graph[e].size()};
34         edge backward={s, -cost, 0, 0, graph[s].size()};
35         if (cost < 0 || ranbefore) needNormalize = true;
36         graph[s].emplace_back(forward);
37         graph[e].emplace_back(backward);
38     }
39     bool normalize(int s) {
40         auto infinite_cost = numeric_limits<cost_t>::max();
41         vector<cost_t> dist(n, infinite_cost);
42         dist[s] = 0;
43         queue<int> q;
44         vector<int> v(n), relax_count(n);
45         v[s] = 1; q.push(s);
46         while (!q.empty()) {
47             int cur = q.front();
48             v[cur] = 0; q.pop();
49             if (++relax_count[cur] >= n) return false;
50             for (const auto &e : graph[cur]) {
51                 if (iszerocap(e.residual_capacity)) continue;
52                 auto next = e.target;
53                 auto ncost = dist[cur] + e.cost;
54                 if (dist[next] > ncost) {
55                     dist[next] = ncost;
56                     if (v[next]) continue;
57                     v[next] = 1; q.push(next);
58                 }
59             }
60         }
61         for (int i = 0; i < n; i++) pi[i] = dist[i];
62         return true;
63     }
64
65     pair<cost_t, cap_t> AugmentShortest(int s, int e, cap_t flow_limit) {
66         auto infinite_cost = numeric_limits<cost_t>::max();
67         auto infinite_flow = numeric_limits<cap_t>::max();
68         typedef pair<cost_t, int> pq_t;
69         priority_queue<pq_t, vector<pq_t>, greater<pq_t>> pq;
70         vector<pair<cost_t, cap_t>> dist(n, make_pair(infinite_cost, 0));
71         vector<int> from(n, -1), v(n);
```

```
72          if (needNormalize || (ranbefore && lastStart != s))
73              normalize(s);
74          ranbefore = true;
75          lastStart = s;
76
77
78          dist[s] = pair<cost_t, cap_t>(0, infinite_flow);
79          pq.emplace(dist[s].first, s);
80          while (!pq.empty()) {
81              auto cur = pq.top().second; pq.pop();
82              if (v[cur]) continue;
83              v[cur] = 1;
84              if (cur == e) continue;
85              for (const auto &e : graph[cur]) {
86                  auto next = e.target;
87                  if (v[next]) continue;
88                  if (iszerocap(e.residual_capacity)) continue;
89                  auto ncost = dist[cur].first + e.cost - pi[next] + pi[cur];
90                  auto nflow = min(dist[cur].second, e.residual_capacity);
91                  if (dist[next].first <= ncost) continue;
92                  dist[next] = make_pair(ncost, nflow);
93                  from[next] = e.revid;
94                  pq.emplace(dist[next].first, next);
95              }
96          }
97          /** augment the shortest path **/
98          auto p = e;
99          auto pathcost = dist[p].first + pi[p] - pi[s];
100         auto flow = dist[p].second;
101         if (iszerocap(flow)|| (flow_limit <= 0 && pathcost >= 0)) return pair<
               cost_t, cap_t>(0, 0);
102         if (flow_limit > 0) flow = min(flow, flow_limit);
103         /* update potential */
104         for (int i = 0; i < n; i++) {
105             if (iszerocap(dist[i].second)) continue;
106             pi[i] += dist[i].first;
107         }
108         while (from[p] != -1) {
109             auto nedge = from[p];
110             auto np = graph[p][nedge].target;
111             auto fedge = graph[p][nedge].revid;
112             graph[p][nedge].residual_capacity += flow;
113             graph[np][fedge].residual_capacity -= flow;
114             p = np;
115         }
116         return make_pair(pathcost * flow, flow);
117     }
118
119     pair<cost_t,cap_t> solve(int s, int e, cap_t flow_minimum = numeric_limits<
           cap_t>::max()) {
120         cost_t total_cost = 0;
121         cap_t total_flow = 0;
122         for(;;) {
123             auto res = AugmentShortest(s, e, flow_minimum - total_flow);
124             if (res.second <= 0) break;
125             total_cost += res.first;
126             total_flow += res.second;
127         }
128         return make_pair(total_cost, total_flow);
129     }
130 };
```

## 5.12  General Min-cut (Stoer-Wagner)

```
1  // implementation of Stoer-Wagner algorithm
2  // O(V^3)
3  //usage
4  // MinCut mc;
5  // mc.init(n);
6  // for (each edge) mc.addEdge(a,b,weight);
7  // mincut = mc.solve();
8  // mc.cut = {0,1}^n describing which side the vertex belongs to.
9  struct MinCutMatrix
10 {
11     typedef int cap_t;
12     int n;
13     vector<vector<cap_t>> graph;
14
15     void init(int _n) {
16         n = _n;
17         graph = vector<vector<cap_t>>(n, vector<cap_t>(n, 0));
18     }
19     void addEdge(int a, int b, cap_t w) {
20         if (a == b) return;
21         graph[a][b] += w;
22         graph[b][a] += w;
23     }
24
25     pair<cap_t, pair<int, int>> stMinCut(vector<int> &active) {
26         vector<cap_t> key(n);
27         vector<int> v(n);
28         int s = -1, t = -1;
29         for (int i = 0; i < active.size(); i++) {
30             cap_t maxv = -1;
31             int cur = -1;
32             for (auto j : active) {
33                 if (v[j] == 0 && maxv < key[j]) {
34                     maxv = key[j];
35                     cur = j;
36                 }
37             }
38             t = s; s = cur;
39             v[cur] = 1;
40             for (auto j : active) key[j] += graph[cur][j];
41         }
42         return make_pair(key[s], make_pair(s, t));
43     }
44
45     vector<int> cut;
```

```
46
47    cap_t solve() {
48        cap_t res = numeric_limits<cap_t>::max();
49        vector<vector<int>> grps;
50        vector<int> active;
51        cut.resize(n);
52        for (int i = 0; i < n; i++) grps.emplace_back(1, i);
53        for (int i = 0; i < n; i++) active.push_back(i);
54        while (active.size() >= 2) {
55            auto stcut = stMinCut(active);
56            if (stcut.first < res) {
57                res = stcut.first;
58                fill(cut.begin(), cut.end(), 0);
59                for (auto v : grps[stcut.second.first]) cut[v] = 1;
60            }
61
62            int s = stcut.second.first, t = stcut.second.second;
63            if (grps[s].size() < grps[t].size()) swap(s, t);
64
65            active.erase(find(active.begin(), active.end(), t));
66            grps[s].insert(grps[s].end(), grps[t].begin(), grps[t].end());
67            for (int i = 0; i < n; i++) { graph[i][s] += graph[i][t]; graph[i][t
                 ] = 0; }
68            for (int i = 0; i < n; i++) { graph[s][i] += graph[t][i]; graph[t][i
                 ] = 0; }
69            graph[s][s] = 0;
70        }
71        return res;
72    }
73 };
```

## 5.13   Hungarian Algorithm

```
1  namespace hung {
2      /*
3       * alternative cost_t example
4       *
5       * typedef pair<int,int> cost_t;
6       * cost_t MAX_COST = make_pair(2,0);
7       * pair<int,int> &operator += (pair<int,int> &a, const pair<int,int> &b) {
8       *     a.first += b.first; a.second += b.second;
9       *     return a;
10      * }
11      * pair<int,int> &operator -= (pair<int,int> &a, const pair<int,int> &b) {
12      *     a.first -= b.first; a.second -= b.second;
13      *     return a;
14      * }
15      */
16     typedef int cost_t;
17     cost_t MAX_COST = numeric_limits<cost_t>::max() / 2;
18     // input: n, dat(which is NOT const)
19     // output: call solve(), match, matched
20     // minimum matching 계산이다.
21     const int MAX_N = 500;
```

```
22
23     int n, match[MAX_N], matched[MAX_N];
24     cost_t dat[MAX_N][MAX_N];
25     int q[MAX_N], v[MAX_N], vcnt;
26     int f[MAX_N], reach[MAX_N], reach2[MAX_N], rcnt;
27
28     int augment(int r) {
29         int h, t = 0;
30         v[r] = ++vcnt;
31         q[t++] = r;
32         for (h = 0; h<t; h++) {
33             int qh = q[h];
34             for (int j = 0; j<n; j++) {
35                 if (dat[qh][j] != 0) continue;
36                 int next = matched[j];
37                 if (next == -1) {
38                     for (;;) {
39                         int org = match[qh];
40                         match[qh] = j; matched[j] = qh;
41                         if (qh == r) return 1;
42                         qh = f[qh]; j = org;
43                     }
44                 }
45                 else if (v[next] != vcnt) {
46                     v[next] = vcnt, f[next] = qh, q[t++] = next;
47                 }
48             }
49         }
50         for (int i = 0; i<n; i++)
51             if (v[i] == vcnt) {
52                 reach[i] = rcnt;
53                 if (i != r) reach2[match[i]] = rcnt;
54             }
55         return 0;
56     }
57
58     cost_t solve() {
59         cost_t ans = 0;
60         for (int i = 0; i<n; i++) match[i] = matched[i] = -1;
61         for (int i = 0; i<n; i++) {
62             cost_t minv = *min_element(dat[i], dat[i] + n);
63             for (int j = 0; j < n; j++) dat[i][j] -= minv;
64             ans += minv;
65             minv = dat[0][i];
66             for (int j = 1; j<n; j++) minv = min(minv, dat[j][i]);
67             for (int j = 0; j<n; j++) dat[j][i] -= minv;
68             ans += minv;
69         }
70         for (;;) {
71             ++rcnt;
72             bool needMore = false;
73             for (int i = 0; i<n; i++) {
74                 if (match[i] >= 0) continue;
75                 if (!augment(i)) needMore = true;
76             }
```

```
77                if (!needMore) break;
78                cost_t minv = MAX_COST;
79                for (int i = 0; i<n; i++) {
80                    if (reach[i] != rcnt) continue;
81                    for (int j = 0; j<n; j++) {
82                        if (reach2[j] == rcnt) continue;
83                        minv = min(minv, dat[i][j]);
84                    }
85                }
86                for (int i = 0; i<n; i++) {
87                    if (match[i]<0) ans += minv;
88                    for (int j = 0; j<n; j++) {
89                        if (reach[i] != rcnt) dat[i][j] += minv;
90                        if (reach2[j] != rcnt) dat[i][j] -= minv;
91                    }
92                }
93            }
94        return ans;
95    }
96 }
```

# 6 Geometry

## 6.1 Basic Operations

```
1  const double eps = 1e-9;
2
3  inline int diff(double lhs, double rhs) {
4      if (lhs - eps < rhs && rhs < lhs + eps) return 0;
5      return (lhs < rhs) ? -1 : 1;
6  }
7
8  inline bool is_between(double check, double a, double b) {
9      if (a < b)
10         return (a - eps < check && check < b + eps);
11     else
12         return (b - eps < check && check < a + eps);
13 }
14
15 struct Point {
16     double x, y;
17     bool operator==(const Point& rhs) const {
18         return diff(x, rhs.x) == 0 && diff(y, rhs.y) == 0;
19     }
20     Point operator+(const Point& rhs) const {
21         return Point{ x + rhs.x, y + rhs.y };
22     }
23     Point operator-(const Point& rhs) const {
24         return Point{ x - rhs.x, y - rhs.y };
25     }
26     Point operator*(double t) const {
27         return Point{ x * t, y * t };
28     }
```

```
29 };
30
31 struct Circle {
32     Point center;
33     double r;
34 };
35
36 struct Line {
37     Point pos, dir;
38 };
39
40 inline double inner(const Point& a, const Point& b) {
41     return a.x * b.x + a.y * b.y;
42 }
43
44 inline double outer(const Point& a, const Point& b) {
45     return a.x * b.y - a.y * b.x;
46 }
47
48 inline int ccw_line(const Line& line, const Point& point) {
49     return diff(outer(line.dir, point - line.pos), 0);
50 }
51
52 inline int ccw(const Point& a, const Point& b, const Point& c) {
53     return diff(outer(b - a, c - a), 0);
54 }
55
56 inline double dist(const Point& a, const Point& b) {
57     return sqrt(inner(a - b, a - b));
58 }
59
60 inline double dist2(const Point &a, const Point &b) {
61     return inner(a - b, a - b);
62 }
63
64 inline double dist(const Line& line, const Point& point, bool segment = false) {
65     double c1 = inner(point - line.pos, line.dir);
66     if (segment && diff(c1, 0) <= 0) return dist(line.pos, point);
67     double c2 = inner(line.dir, line.dir);
68     if (segment && diff(c2, c1) <= 0) return dist(line.pos + line.dir, point);
69     return dist(line.pos + line.dir * (c1 / c2), point);
70 }
71
72 bool get_cross(const Line& a, const Line& b, Point& ret) {
73     double mdet = outer(b.dir, a.dir);
74     if (diff(mdet, 0) == 0) return false;
75     double t2 = outer(a.dir, b.pos - a.pos) / mdet;
76     ret = b.pos + b.dir * t2;
77     return true;
78 }
79
80 bool get_segment_cross(const Line& a, const Line& b, Point& ret) {
81     double mdet = outer(b.dir, a.dir);
82     if (diff(mdet, 0) == 0) return false;
83     double t1 = -outer(b.pos - a.pos, b.dir) / mdet;
```

```
 84        double t2 = outer(a.dir, b.pos - a.pos) / mdet;
 85        if (!is_between(t1, 0, 1) || !is_between(t2, 0, 1)) return false;
 86        ret = b.pos + b.dir * t2;
 87        return true;
 88 }
 89
 90 Point inner_center(const Point &a, const Point &b, const Point &c) {
 91        double wa = dist(b, c), wb = dist(c, a), wc = dist(a, b);
 92        double w = wa + wb + wc;
 93        return Point{ (wa * a.x + wb * b.x + wc * c.x) / w, (wa * a.y + wb * b.y +
              wc * c.y) / w };
 94 }
 95
 96 Point outer_center(const Point &a, const Point &b, const Point &c) {
 97        Point d1 = b - a, d2 = c - a;
 98        double area = outer(d1, d2);
 99        double dx = d1.x * d1.x * d2.y - d2.x * d2.x * d1.y
100            + d1.y * d2.y * (d1.y - d2.y);
101        double dy = d1.y * d1.y * d2.x - d2.y * d2.y * d1.x
102            + d1.x * d2.x * (d1.x - d2.y);
103        return Point{ a.x + dx / area / 2.0, a.y - dy / area / 2.0 };
104 }
105
106 vector<Point> circle_line(const Circle& circle, const Line& line) {
107        vector<Point> result;
108        double a = 2 * inner(line.dir, line.dir);
109        double b = 2 * (line.dir.x * (line.pos.x - circle.center.x)
110            + line.dir.y * (line.pos.y - circle.center.y));
111        double c = inner(line.pos - circle.center, line.pos - circle.center)
112            - circle.r * circle.r;
113        double det = b * b - 2 * a * c;
114        int pred = diff(det, 0);
115        if (pred == 0)
116            result.push_back(line.pos + line.dir * (-b / a));
117        else if (pred > 0) {
118            det = sqrt(det);
119            result.push_back(line.pos + line.dir * ((-b + det) / a));
120            result.push_back(line.pos + line.dir * ((-b - det) / a));
121        }
122        return result;
123 }
124
125 vector<Point> circle_circle(const Circle& a, const Circle& b) {
126        vector<Point> result;
127        int pred = diff(dist(a.center, b.center), a.r + b.r);
128        if (pred > 0) return result;
129        if (pred == 0) {
130            result.push_back((a.center * b.r + b.center * a.r) * (1 / (a.r + b.r)));
131            return result;
132        }
133        double aa = a.center.x * a.center.x + a.center.y * a.center.y - a.r * a.r;
134        double bb = b.center.x * b.center.x + b.center.y * b.center.y - b.r * b.r;
135        double tmp = (bb - aa) / 2.0;
136        Point cdiff = b.center - a.center;
137        if (diff(cdiff.x, 0) == 0) {
138            if (diff(cdiff.y, 0) == 0)
139                return result; // if (diff(a.r, b.r) == 0): same circle
140            return circle_line(a, Line{ Point{ 0, tmp / cdiff.y }, Point{ 1, 0 } });
141        }
142        return circle_line(a,
143            Line{ Point{ tmp / cdiff.x, 0 }, Point{ -cdiff.y, cdiff.x } });
144 }
145
146 Circle circle_from_3pts(const Point& a, const Point& b, const Point& c) {
147        Point ba = b - a, cb = c - b;
148        Line p{ (a + b) * 0.5, Point{ ba.y, -ba.x } };
149        Line q{ (b + c) * 0.5, Point{ cb.y, -cb.x } };
150        Circle circle;
151        if (!get_cross(p, q, circle.center))
152            circle.r = -1;
153        else
154            circle.r = dist(circle.center, a);
155        return circle;
156 }
157
158 Circle circle_from_2pts_rad(const Point& a, const Point& b, double r) {
159        double det = r * r / dist2(a, b) - 0.25;
160        Circle circle;
161        if (det < 0)
162            circle.r = -1;
163        else {
164            double h = sqrt(det);
165            // center is to the left of a->b
166            circle.center = (a + b) * 0.5 + Point{ a.y - b.y, b.x - a.x } * h;
167            circle.r = r;
168        }
169        return circle;
170 }
```

## 6.2 Compare angles

## 6.3 Convex Hull

```
 1 // find convex hull
 2 // O(n*logn)
 3 vector<Point> convex_hull(vector<Point>& dat) {
 4        if (dat.size() <= 3) return dat;
 5        vector<Point> upper, lower;
 6        sort(dat.begin(), dat.end(), [](const Point& a, const Point& b) {
 7            return (a.x == b.x) ? a.y < b.y : a.x < b.x;
 8        });
 9        for (const auto& p : dat) {
10            while (upper.size() >= 2 && ccw(*++upper.rbegin(), *upper.rbegin(), p)
                   >= 0) upper.pop_back();
11            while (lower.size() >= 2 && ccw(*++lower.rbegin(), *lower.rbegin(), p)
                   <= 0) lower.pop_back();
12            upper.emplace_back(p);
13            lower.emplace_back(p);
```

```
14        }
15        upper.insert(upper.end(), ++lower.rbegin(), --lower.rend());
16        return upper;
17  }
```

### 6.4   Polygon Cut

```
1  // left side of a->b
2  vector<Point> cut_polygon(const vector<Point>& polygon, Line line) {
3      if (!polygon.size()) return polygon;
4      typedef vector<Point>::const_iterator piter;
5      piter la, lan, fi, fip, i, j;
6      la = lan = fi = fip = polygon.end();
7      i = polygon.end() - 1;
8      bool lastin = diff(ccw_line(line, polygon[polygon.size() - 1]), 0) > 0;
9      for (j = polygon.begin() ; j != polygon.end() ; j++) {
10         bool thisin = diff(ccw_line(line, *j), 0) > 0;
11         if (lastin && !thisin) {
12             la = i;
13             lan = j;
14         }
15         if (!lastin && thisin) {
16             fi = j;
17             fip = i;
18         }
19         i = j;
20         lastin = thisin;
21     }
22     if (fi == polygon.end()) {
23         if (!lastin) return vector<Point>();
24         return polygon;
25     }
26     vector<Point> result;
27     for (i = fi ; i != lan ; i++) {
28         if (i == polygon.end()) {
29             i = polygon.begin();
30             if (i == lan) break;
31         }
32         result.push_back(*i);
33     }
34     Point lc, fc;
35     get_cross(Line{ *la, *lan - *la }, line, lc);
36     get_cross(Line{ *fip, *fi - *fip }, line, fc);
37     result.push_back(lc);
38     if (diff(dist2(lc, fc), 0) != 0) result.push_back(fc);
39     return result;
40  }
```

### 6.5   Pick's theorem

격자점으로 구성된 simple polygon이 주어짐. $i$는 polygon 내부의 격자점 수, $b$는 polygon 선분 위 격자점 수, $A$는 polygon의 넓이라고 할 때, 다음과 같은 식이 성립한다.

$$A = i + \frac{b}{2} - 1$$

## 7   String

### 7.1   KMP

```
1  typedef vector<int> seq_t;
2
3  void calculate_pi(vector<int>& pi, const seq_t& str) {
4      pi[0] = -1;
5      for (int i = 1, j = -1; i < str.size(); i++) {
6          while (j >= 0 && str[i] != str[j + 1]) j = pi[j];
7          if (str[i] == str[j + 1])
8              pi[i] = ++j;
9          else
10             pi[i] = -1;
11     }
12  }
13
14  // returns all positions matched
15  // O(|text|+|pattern|)
16  vector<int> kmp(const seq_t& text, const seq_t& pattern) {
17      vector<int> pi(pattern.size()), ans;
18      if (pattern.size() == 0) return ans;
19      calculate_pi(pi, pattern);
20      for (int i = 0, j = -1; i < text.size(); i++) {
21          while (j >= 0 && text[i] != pattern[j + 1]) j = pi[j];
22          if (text[i] == pattern[j + 1]) {
23              j++;
24              if (j + 1 == pattern.size()) {
25                  ans.push_back(i - j);
26                  j = pi[j];
27              }
28          }
29      }
30      return ans;
31  }
```

### 7.2   Aho-Corasick

```
1  #include <algorithm>
2  #include <vector>
3  #include <queue>
4  using namespace std;
5
6  struct AhoCorasick
7  {
8      const int alphabet;
9      struct node {
10         node() {}
```

```
11          explicit node(int alphabet) : next(alphabet) {}
12          vector<int> next, report;
13          int back = 0, output_link = 0;
14      };
15      int maxid = 0;
16      vector<node> dfa;
17      explicit AhoCorasick(int alphabet) : alphabet(alphabet), dfa(1, node(
          alphabet)) { }
18      template<typename InIt, typename Fn> void add(int id, InIt first, InIt last,
          Fn func) {
19          int cur = 0;
20          for ( ; first != last; ++first) {
21              auto s = func(*first);
22              if (auto next = dfa[cur].next[s]) cur = next;
23              else {
24                  cur = dfa[cur].next[s] = (int)dfa.size();
25                  dfa.emplace_back(alphabet);
26              }
27          }
28          dfa[cur].report.push_back(id);
29          maxid = max(maxid, id);
30      }
31      void build() {
32          queue<int> q;
33          vector<char> visit(dfa.size());
34          visit[0] = 1;
35          q.push(0);
36          while(!q.empty()) {
37              auto cur = q.front(); q.pop();
38              dfa[cur].output_link = dfa[cur].back;
39              if (dfa[dfa[cur].back].report.empty())
40                  dfa[cur].output_link = dfa[dfa[cur].back].output_link;
41              for (int s = 0; s < alphabet; s++) {
42                  auto &next = dfa[cur].next[s];
43                  if (next == 0) next = dfa[dfa[cur].back].next[s];
44                  if (visit[next]) continue;
45                  if (cur) dfa[next].back = dfa[dfa[cur].back].next[s];
46                  visit[next] = 1;
47                  q.push(next);
48              }
49          }
50      }
51      template<typename InIt, typename Fn> vector<int> countMatch(InIt first, InIt
          last, Fn func) {
52          int cur = 0;
53          vector<int> ret(maxid+1);
54          for (; first != last; ++first) {
55              cur = dfa[cur].next[func(*first)];
56              for (int p = cur; p; p = dfa[p].output_link)
57                  for (auto id : dfa[p].report) ret[id]++;
58          }
59          return ret;
60      }
61  };
```

## 7.3  Suffix Array with LCP

```
1  typedef char T;
2
3  // calculates suffix array.
4  // O(n*logn)
5  vector<int> suffix_array(const vector<T>& in) {
6      int n = (int)in.size(), c = 0;
7      vector<int> temp(n), pos2bckt(n), bckt(n), bpos(n), out(n);
8      for (int i = 0; i < n; i++) out[i] = i;
9      sort(out.begin(), out.end(), [&](int a, int b) { return in[a] < in[b]; });
10     for (int i = 0; i < n; i++) {
11         bckt[i] = c;
12         if (i + 1 == n || in[out[i]] != in[out[i + 1]]) c++;
13     }
14     for (int h = 1; h < n && c < n; h <<= 1) {
15         for (int i = 0; i < n; i++) pos2bckt[out[i]] = bckt[i];
16         for (int i = n - 1; i >= 0; i--) bpos[bckt[i]] = i;
17         for (int i = 0; i < n; i++)
18             if (out[i] >= n - h) temp[bpos[bckt[i]]++] = out[i];
19         for (int i = 0; i < n; i++)
20             if (out[i] >= h) temp[bpos[pos2bckt[out[i] - h]]++] = out[i] - h;
21         c = 0;
22         for (int i = 0; i + 1 < n; i++) {
23             int a = (bckt[i] != bckt[i + 1]) || (temp[i] >= n - h)
24                      || (pos2bckt[temp[i + 1] + h] != pos2bckt[temp[i] + h]);
25             bckt[i] = c;
26             c += a;
27         }
28         bckt[n - 1] = c++;
29         temp.swap(out);
30     }
31     return out;
32 }
33
34 // calculates lcp array. it needs suffix array & original sequence.
35 // O(n)
36 vector<int> lcp(const vector<T>& in, const vector<int>& sa) {
37     int n = (int)in.size();
38     if (n == 0) return vector<int>();
39     vector<int> rank(n), height(n - 1);
40     for (int i = 0; i < n; i++) rank[sa[i]] = i;
41     for (int i = 0, h = 0; i < n; i++) {
42         if (rank[i] == 0) continue;
43         int j = sa[rank[i] - 1];
44         while (i + h < n && j + h < n && in[i + h] == in[j + h]) h++;
45         height[rank[i] - 1] = h;
46         if (h > 0) h--;
47     }
48     return height;
49 }
```

### 7.4   Suffix Tree

### 7.5   Manacher's Algorithm

```cpp
// find longest palindromic span for each element in str
// O(|str|)
void manacher(const string& str, int plen[]) {
    int r = -1, p = -1;
    for (int i = 0; i < str.length(); ++i) {
        if (i <= r)
            plen[i] = min((2 * p - i >= 0) ? plen[2 * p - i] : 0, r - i);
        else
            plen[i] = 0;
        while (i - plen[i] - 1 >= 0 && i + plen[i] + 1 < str.length()
                && str[i - plen[i] - 1] == str[i + plen[i] + 1]) {
            plen[i] += 1;
        }
        if (i + plen[i] > r) {
            r = i + plen[i];
            p = i;
        }
    }
}
```

## 8   Miscellaneous

### 8.1   Fast I/O

```cpp
namespace fio {
    const int BSIZE = 524288;
    char buffer[BSIZE];
    int p = BSIZE;
    inline char readChar() {
        if(p == BSIZE) {
            fread(buffer, 1, BSIZE, stdin);
            p = 0;
        }
        return buffer[p++];
    }
    int readInt() {
        char c = readChar();
        while ((c < '0' || c > '9') && c != '-') {
            c = readChar();
        }
        int ret = 0; bool neg = c == '-';
        if (neg) c = readChar();
        while (c >= '0' && c <= '9') {
            ret = ret * 10 + c - '0';
            c = readChar();
        }
        return neg ? -ret : ret;
```

```
    }
}
```

### 8.2   Magic Numbers

소수 : $10\,007$ , $10\,009$ , $10\,111$ , $31\,567$ , $70\,001$ , $1\,000\,003$ , $1\,000\,033$ , $4\,000\,037$ , $1\,000\,000\,007$ , $1\,000\,000\,009$

### 8.3   Java Examples

```java
import java.util.Scanner;

public class example
{
    public static void main(String[] args)
    {
        Scanner in = new Scanner(System.in);
        int T = in.nextInt();
        while (T --> 0)
        {
            String str = in.next();
            if (str.matches("[A-F]?A+F+C+[A-F]?"))
                System.out.println("Infected!");
            else
                System.out.println("Good");
        }
    }
}
```